

Tests de bout en bout en shell avec “bats”

pour CLI, containers et clusters



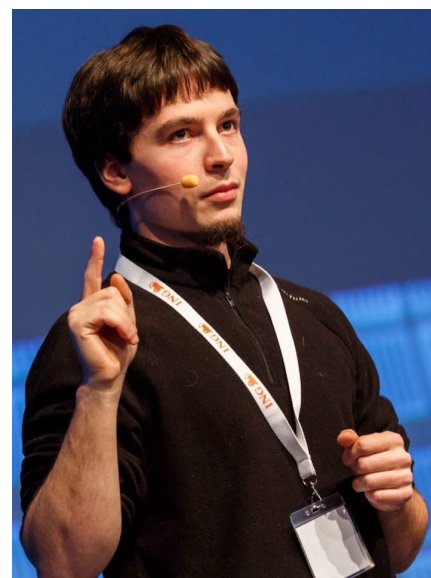
Présentation disponible à l'URL: <https://dduportal.github.io/slides/gdg-lille-2019>

Comment utiliser cette présentation ?

- Pour naviguer, utilisez les flèches en bas à droite (ou celles de votre clavier)
 - Gauche/Droite: changer de chapitre
 - Haut/Bas: naviguer dans un chapitre
- Pour avoir une vue globale : utiliser la touche "o" (pour "**O**verview")
- Pour voir les notes de l'auteur : utilisez la touche "s" (pour "**S**peaker notes")

Whoami

- Damien DUPORTAL:
 - Træfik's Developer 🥑 Advocate @ Containous and Freelancer
 - Former Training Engineer @ CloudBees
- 🐦 @DamienDuportal
- 🐙 dduportal



Menu

- Tests de bout en bout: Kezako ?
- Tester une CLI
- Tester des conteneurs
- Tester une application complexe

Tester une CLI

Problématique

- Je fournis une CLI avec un README
- Comment s'assurer que la documentation:
 - Est à jour ?
 - Est fonctionnelle ?

Idée

- Test "E2E" pour jouer les scénarios écrits (ref. AsciiDoctor)
- Jouer une command == automatiser du shell

bats

Bats is a TAP-compliant testing framework for Bash. It provides a simple way to verify that the UNIX programs you write behave as expected.

bats on 

Exemple

```
#!/usr/bin/env bats
# Fichier "test-docker.bats"

@test "Binary docker is on the current PATH" {
    command -v docker
}

DOCKER_VERSION=18.09.3
@test "docker version is ${DOCKER_VERSION}" {
    docker -v | grep "${DOCKER_VERSION}"
}
```

Execution

```
$ bats test-docker.bats
✓ Binary docker is on the current PATH
✓ docker version is 18.09.3

2 tests, 0 failures
```

TAP: Test Anything Protocol

- Bats permet d'exporter en format TAP
 - Parsable par une machine
 - Vient du monde Perl, mais beaucoup de modules: (exemple: Jenkins TAP module)

```
bats --tap addition.bats
1..2
ok 1 Binary docker is on the current PATH
ok 2 docker version is 18.09.3
```

Facilitées : run

- `run <COMMAND>` pour tester le exit code ou l'output

```
@test "ls sur un fichier non existant a une erreur 1" {  
  run ls -l /nonexistent_filename  
  [ "$status" -eq 1 ]  
  [ "$output" = "ls:/nonexistent_filename: No such file or directory"  
}
```

Facilitées : load

- `load custom_helper` pour réutiliser du code

```
#!/bin/bash
# fichier "custom_helper.bash"

say_hello() {
    echo "Hello"
}
```

```
#!/bin/bats
load custom_helper

@test "Dire bonjour" {
    say_hello
}
```

Facilitées : Hooks

- `setup()` et `teardown()` sont exécutés (respectivement) avant et après chaque test

```
setup() {  
    eval $(minikube env)  
}  
teardown() {  
    unset DOCKER_HOST  
}  
  
@test "docker avec minikube" {  
    # setup has been run  
    docker ps # Using minikube  
    # teardown will be run  
}
```

Facilitées : Variables

- `$BATS_TEST_FILENAME`
- `$BATS_TEST_DIRNAME`
- `$BATS_TEST_NAMES`
- `$BATS_TEST_NAME`
- `$BATS_TEST_DESCRIPTION`
- `$BATS_TEST_NUMBER`
- `$BATS_TMPDIR`

Tester des conteneurs / images

bats pour les conteneurs

- Exemple avec l'image Docker Jenkins

```
#!/usr/bin/env bats
load 'test_helper/bats-support/load'
load 'test_helper/bats-assert/load'
load test_helpers

...

@test "create test container" {
    docker run -d -e JAVA_OPTS="-Duser.timezone=Europe/Madrid -Dhudson.model.DirectoryB
}

@test "test container is running" {
    sleep 1 # give time to eventually fail to initialize
    retry 3 1 assert "true" docker inspect -f {{.State.Running}} $SUT_CONTAINER
}
```

Problèmes avec bats

- Dernier commit : 19 février 2016...
- Erreurs difficile à comprendre
- Shell
- Installation

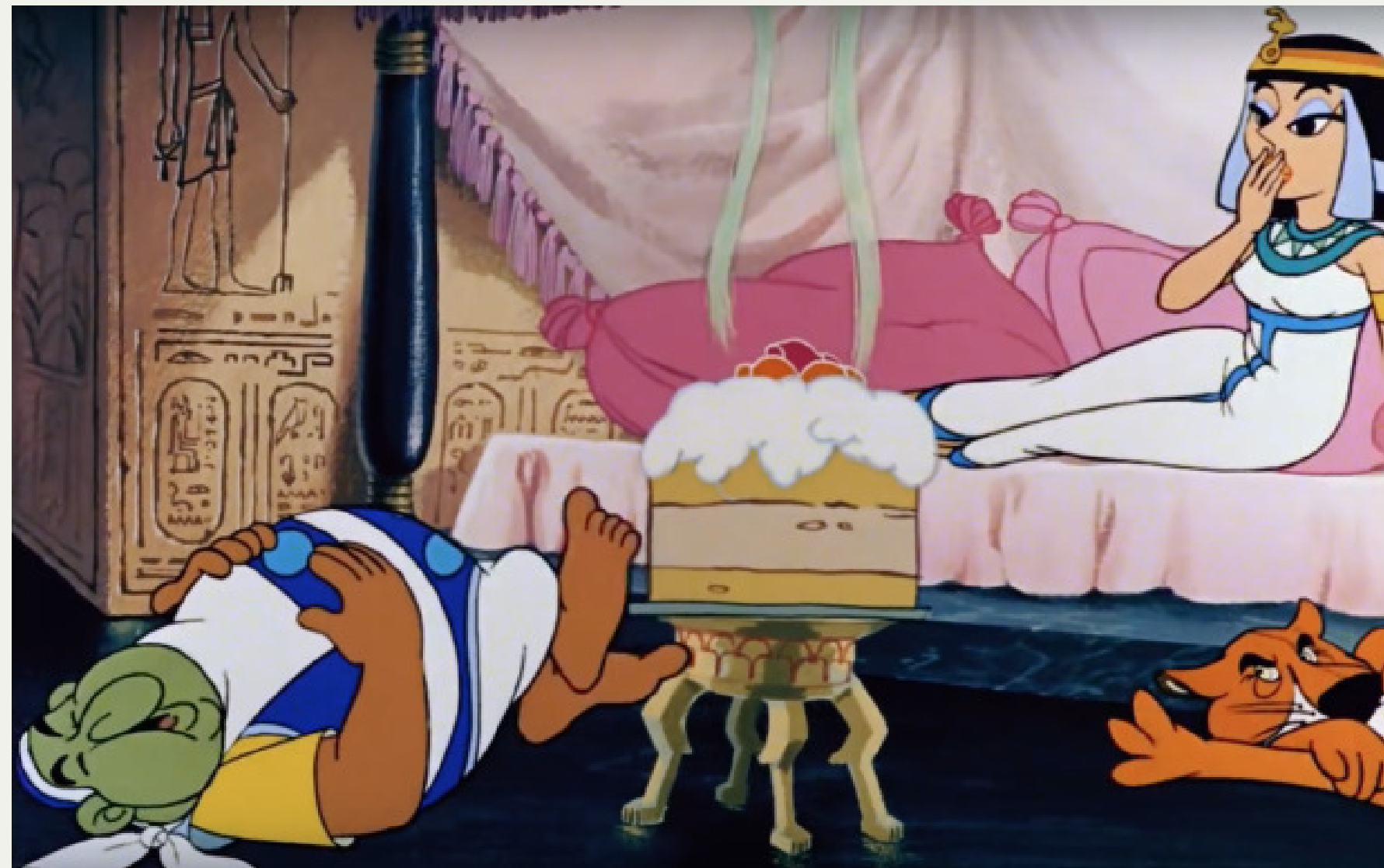
Communauté

- Fork : <https://github.com/bats-core/bats-core>
 - Communauté plus disponible (vs. 1 maintainer)
 - Documentation améliorée
 - Installation avec npm, brew, etc.
- Helper Libraries : <https://github.com/ztombol/bats-docs>
 - Facilite certaines manipulation en shell
 - Propose des assertions, améliorant de fait les erreurs outputs

Packaging

- Objectif : Packager de manière portable
- Dénominateur commun : npm
 - Windows ? Sécurité ?
- Solution : Fournir `bat s` dans un conteneur Docker
 - Les "helpers" librairies, CLIs et configurations sont des dépendances fixées

Gouter sa propre nourriture



Testons l'image Docker bats ... avec bats!

Docker Multi-Stage

```
FROM node:alpine as dependencies-solver
COPY package*.json /bats/
WORKDIR /bats
RUN npm install

FROM alpine:3.9
ENV BATS_HELPERS_DIR=/opt/bats-helpers
COPY --from=dependencies-solver /bats/node_modules/bats /opt/bats
COPY --from=dependencies-solver /bats/node_modules/bats-support /opt/bats-helpers/bats-
COPY --from=dependencies-solver /bats/node_modules/bats-file /opt/bats-helpers/bats-fil
COPY --from=dependencies-solver /bats/node_modules/bats-assert /opt/bats-helpers/bats-a
RUN apk add --no-cache bash
WORKDIR /tests
ENTRYPOINT ["/sbin/bats"]
CMD [ "-v" ]
```

Self-Testing

```
# docker_helper.bash
...
run_command_with_docker() {
    docker run --rm -t ${CUSTOM_DOCKER_RUN_OPTS} \
        "${DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_TAG}" "$@"
}

# test.bats
loader docker_helper

@test "With no cmd/args, the image return Bats version" {
    run_command_with_docker | grep "Bats" | grep "${BATS_VERSION}"
}
```

Résultat

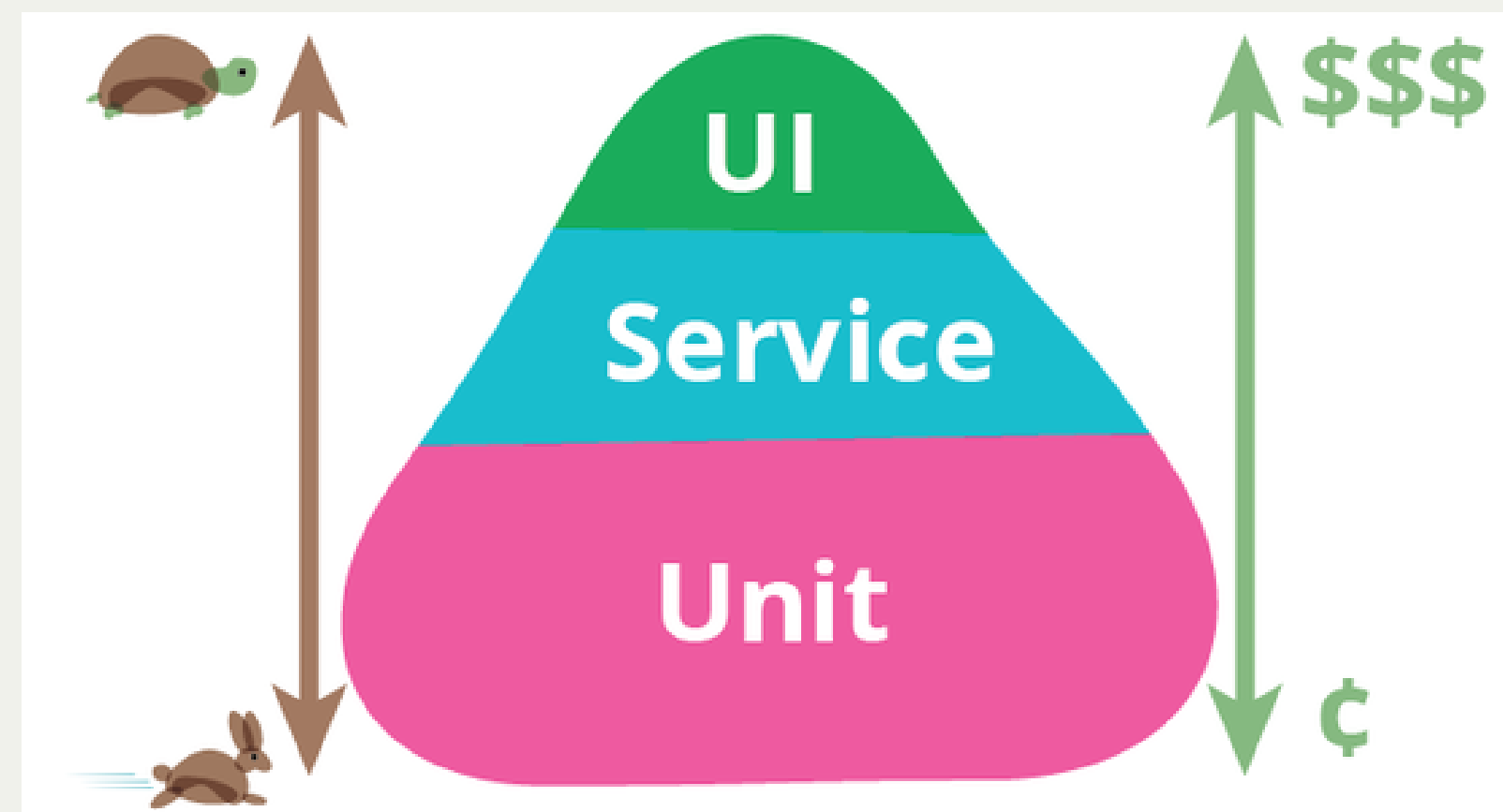
- Facilite la contribution : exemple sur l'image Docker de AsciiDoctor
- "Documentation" automatisée
- Meilleure processus de pensée pour élaborer les changements à plusieurs

Tests de bout en bout: Kezako ?

Tests de bout en bout

- Objet: tester des scénarios d'utilisation de l'application du point de vue de l'utilisateur final
- Usages: BDD ("Behaviour Driven Development") et non régression

Pyramide des tests



Problématiques des test bout en bout

- Complexité
- Lents à exécuter
- Facilement Non-Déterministes (irrégularités)
- Arrivent tard dans le pipeline

Bonne pratiques

- Usage orienté "régression"
- Maniaque sur la qualité
- Lors de l'apparition d'un échec de test:
 - Mettre en exergue le bug au niveau de test unitaire
 - Corriger le problème
- Voir le "bout en bout" comme une seconde ligne ("Nice to have")

Vous ne pouvez pas éviter

- Les automates à états finis
 - "Retries", breaking pattern
- L'asynchronisme
- La sécurisation des credentials
 - Tokens
 - Système externes
- La frustration d'un test à mettre en quarantaine

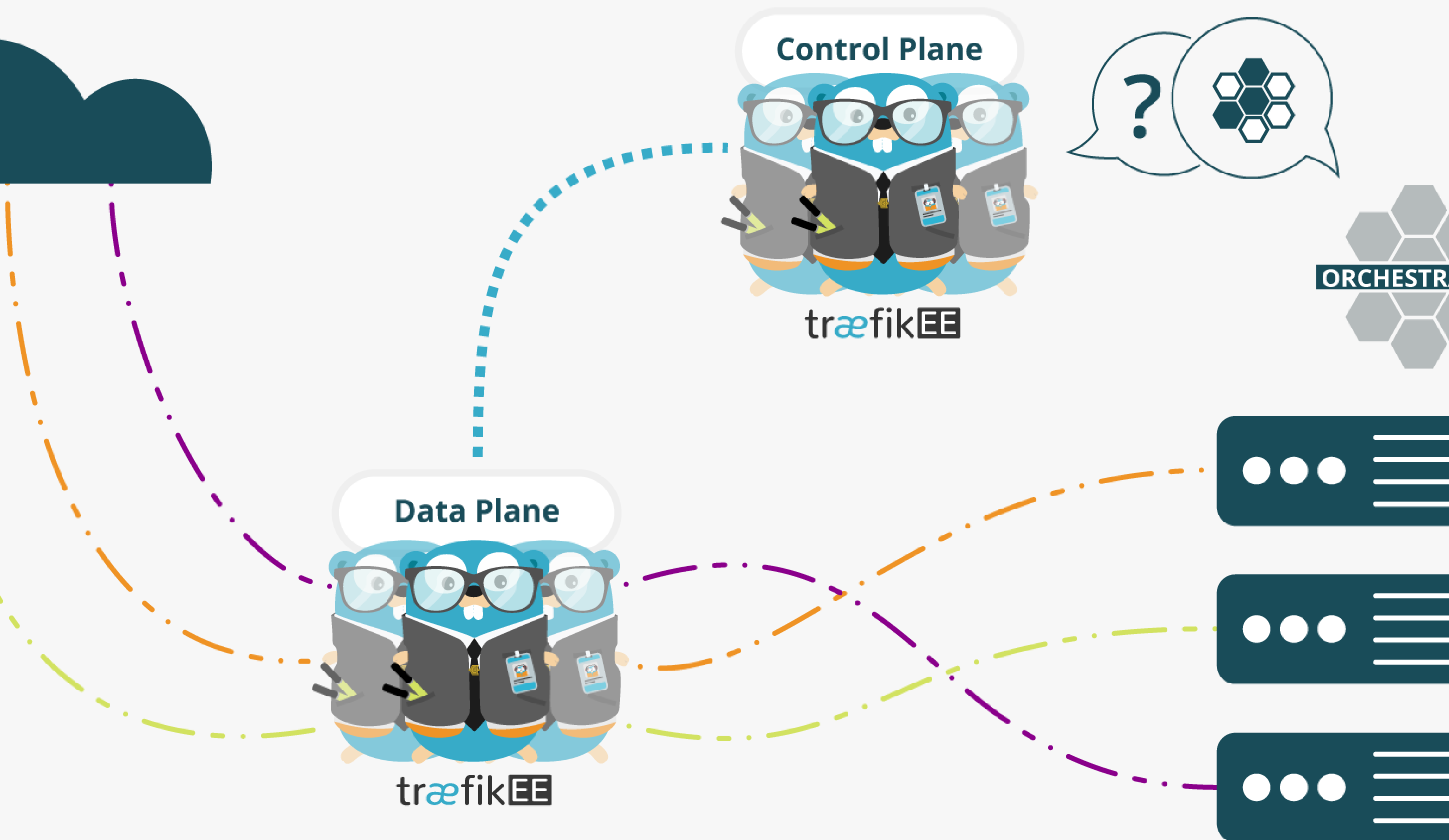
Bibliographie

- (EN) <https://martinfowler.com/bliki/BroadStackTest.html>
- (EN) <https://www.softwaretestinghelp.com/what-is-end-to-end-testing/>
- (FR) <https://blog.testingdigital.com/quest-test-de-bout-bout-end-to-end-1288>
- (FR) <http://douche.name/blog/nomenclature-des-tests-logiciels/>


Tester une application complexe

INTERNET


TO YOUR INFRA



KinD

- Kubernetes in Docker : KinD
 - SIG Kubernetes
- "New kid in town": k3s, de rancher k3s sur  [GitHub](#)
 - Très rapide !

SinD

- Swarm in Docker: SinD sur 
 - Un cluster swarm isolé en ~3-4s
 - On travaille sur du "DockerEE in Docker"!

Isolation

- Docker in Docker in Docker
- Niveau 1: Docker non controlable
- Niveau 2: Docker "externe" pour isoler les tests
- Niveau 3: Les Docker (ou runc/cri-o) des clusters Swarm / Kubernetes

Performances

- Pratique de Production: contrôle des ressources (CPU/Mémoire)
 - Déterminisme des comportements lors de la parallélisation
 - Permet de détecter les race conditions plus facilement
- Mais à un moment, contention système
 - Seule solution : \$\$\$

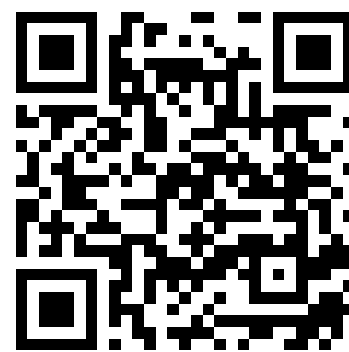
Conclusion

- Un framework complet: "batman" (sera peut être ouvert?)
- Adoption rapide
- Difficultés avec le shell
 - SHELLCHECK
 - Passation de connaissance
- Attention à ne pas faire QUE du bout en bout

Merci !

 @DamienDuportal

 dduportal



<https://dduportal.github.io/slides/gdg-lille-2019>